GBE Version 3
API: New API Specification

Version Number:	2.0.28
Version Date:	20 December 2019

Document Revision History

Version	Date	Comment
2.0.1	10 Nov 2006	Release of the API v2 Info site.
2.0.2	13 Dec 2006	Added ListBlacklistInformation, change PlaceOrdersNoReceipt (added withrawal parameter)
2.0.3	05 Feb 2007	Changed GetPrices, PlaceOrdersNoReceipt, PlaceOrdersWithReceipt, GetEventSubTreeNoSelections, CancelOrders, CancelAllOrders, and CancelAllOrdersOnMarket
2.0.4	14 Feb 2007	Chenged GetPrices (added TotalMatchedAmount), added TimeStamps in all responses, in UpdateOrdersNoReceipt Price and DeltaStake were changed to required fields.
2.0.5	05 Mar 2007	UpdateOrdersNoReceipt: added a new return code 293(InRunningDelayInEffect). GetPrices: added new return code.
2.0.7	16 Apr 2007	Added methods SuspendFromTrading. UnsuspendFromTrading, SuspendOrders, SuspendAllOrdersOnMarket, SuspendAllOrders, RegisterHeartbeat, ChangeHeartbeatRegistration, DeregisterHeartbeat, Pulse.
2.0.10	22 Aug 2007	Added IsFlipped And ShadowSelectionID to GetMarketInformation and GetEventSubTreeWithSelections. Added WantVirtualSelections to GetPrices
2.0.11	18 Sept 2008	Added GetOddsLadder
2.0.12	9 Oct 2008	Added GetCurrentSelectionSequenceNumber Added punterReferenceNumber to GetOrderDetails, ListBootstrapOrders, ListOrdersChangedSince, CancelAllOrders, CancelAllOrdersOnMarket, CancelOrders, SuspendAllOrders, SuspendAllOrdersOnMarket, SuspendOrders, PlaceOrdersWithReceipi, PlaceOrdersNoReceipt
2.0.13	20 Nov 2008	Added postingCategory, handle <Market> to ListAccountPostings
2.0.14	9 Apr 2009	Added output parameters cancelOnInRunning, cancelIfSelectionReset and isCurrentlyInRunning to ListOrdersChangedSince and ListBootstrapOrders Added optional placePayout output parameter to GetEventSubTreeNoSelections, GetEventSubTreeWithSelections, GetMarketInformation and GetPrices ListBootstrapOrders will return suspended orders
2.0.15	10 Mar 2010	Added optional wantPlayMarkets input parameter to ListTopLevelEvents, GetEventSubTreeNoSelections and GetEventSubTreeWithSelections Added isPlayMarket output parameter to GetEventSubTreeNoSelections, GetEventSubTreeWithSelections, GetMarketInformation and GetPrices Added optional wantSettledOrdersOnUnsettledMarkets input parameter to ListBootstrapOrders
2.0.16	9 Jun 2010	Added matchedOrderId output parameter to GetOrderDetails.

Version	Date	Comment
2.0.17	12 Aug 2010	Added optional output parameters wasMake, grossSettlementAmount and orderCommission to GetOrderDetails. Added output parameters totalForSideMakeStake, totalForSideTakeStake, punterCommissionBasis, makeCommissionRate, takeCommissionRate, orderCommission to ListOrdersChangedSince Added output parameters totalForSideMakeStake, totalForSideTakeStake, punterCommissionBasis, makeCommissionRate, takeCommissionRate to ListBootstrapOrders
2.0.18	17 Jan 2011	Removed virtual selection support from GetEventSubTreeWithSelections, GetMarketInformation and GetPrices
2.0.19	29 Jul 2011	Added transactionId to GetAccountPostings output New method added ListAccountPostingsById.
2.0.20	14 Oct 2011	Added wantMarketMatchedAmount, wantSelectionsMatchedAmounts and wantSelectionMatchedDetails input parameters and matchedMarketForStake, matchedMarketAgainstStake, lastMatchedOccurredAt, lastMatchedPrice and lastMatchedForSideAmount output parameters to GetPrices.
2.0.21	21 Jun 2013	Added raceGrade output parameter to GetEventSubTreeNoSelections, GetEventSubTreeWithSelections and GetMarketInformation. Added homeTeamScore, awayTeamScore, scoreType, selectionOpenInterest, marketWinnings, marketPositiveWinnings, lastMatchedAgainstSideAmount, matchedForSideAmountAtSamePrice, matchedAgainstSideAmountAtSamePrice, firstMatchAtSamePriceOccurredAt, numberOrders and numberPunters output parameters to GetPrices.
2.0.22	10 Oct 2014	Added ListSelectionTrades
2.0.23	16 Jun 2015	Added expectedSelectionResetCount and expectedWithdrawalSequenceNumber output parameters to ListOrdersChangedSince and ListBootstrapOrders
2.0.24	19 Jan 2016	Added setToBeSPIfUnmatched input parameter to UpdateOrdersNoReceipt
2.0.25	8 Apr 2016	Added orderFillType and fillOrKillThreshold output parameters to ListBootstrapOrders and ListOrdersChangedSince.
2.0.26	6 May 2016	Added GetSPEnabledMarketsInformation
2.0.27	6 September 2019	Added ListTaggedValues
2.0.28	20 Dcember 2019	Deprecated ChangePassword

Table of Contents

General Points

1. In addition to normal WS return codes, every call explicitly returns a set of GBE-specific return codes. The complete set of GBE-specific return codes that can be returned from any API are listed under the definition of that API.

2. Although these interfaces are defined at this logical level using enumerations (to clearly describe the domain alues that individual parameters can have) there will be no enumerators used in any actual manifestation of these interfaces. This is to facilitate the schema evolution of these interfaces – new domain values can be added while maintaining binary compatibility. All code written to use these interfaces must be aware that new domain values could be added and must be capable of operating elegantly if an un-expected domain value is encountered.

3. The concept of sequence numbers is used heavily in all these interfaces (and in fact using it is the only way to obtain a list of orders). Every time an order is changed in any way the order is assigned a new sequence number. Sequence numbers are guaranteed to increase for every punter over time (ever punter has his how sequence number and sequence numbers can not be compared across punters). An order that has been changed after another order has been changed is guaranteed to have a bigger sequence number than that other order. The sequence number for an order is updated after any change whatsoever is made to the order including changes that were not initiated by the punter, for example when that order gets matched, repriced or cancelled dur to rule 4 withdrawal or settled.

   It is very efficient to identify and retrieve orders for a punter based on the sequence number. The way to use sequence numbers is to record the largest sequence number to date and then poll for orders changed since that sequence number. If a order has been changed it will have a sequence number bigger than that sequence number and so that changed order will be returned. Record the new sequence number that was returned and sepecify that latest sequence number next time you poll for orders.

4. This new API is not a replacement for the existing API. The existing API will continue to exist for the foreseeable future. It is anticipated that a robot will use one API or the other API but will not use a mixture of APIs (although there is no technical reason or restriction why a robot could not use such a mixture). The names of APIs in this new version are intentionally different from similar calls in the existing API to reduce confusion.

5. This API supports suspending orders in addition to cancelling orders. Suspending orders means that the order will not be matched, but the order still exists and funds are still reserved for that order. Suspending orders is very much quicker and more efficient than cancelling orders (as the calculation to determine the new amount of funds to reserve does not need to be performed).

It is probable that at some stage a limit will be imposed on the number of orders that can be specified on PlaceOrdersNoReceipt, PlaceOrdersWithReceipt, UpdateOrdersNoReceipt and the CancelOrders API calls. No limit will be placed on the number or orders that can be specified on the SuspendOrders API calls. It is therefore recommended that robots use the SuspendOrders API calls to immediately ensure that none of their orders will get matched and to then determine update or cancel those orders in batches.

The use of the reciprocal Unsuspend Order call is not intended as a mechanism to present orders for matching i.e. it is not intended as a substitute for Place Order. It is intentionally and substantially restricted in terms of the number of orders that can be unsuspended in any one 60 sec period.

6. The response to every call includes an explicit explicit timestamp, which is the time at which the response was issued by the system. This can be used by clients to determine how current a response is, for example if there are inordinate network delays.

7. Return codes – a number of return codes can be returned by all APIs in addition to the API-specific codes that are explicitly defined for each API. These return codes are:

   - RC000 Success
   - RC001 ResourceError
   - RC002 SystemError
   - RC405 InvalidPassword
   - RC406 PunterIsBlacklisted
   - RC533 PunterNotAuthorisedForAPI

Secure

This group of interfaces deals with all aspects of interacting securely with the API. As a rule these contain all methods that are specific to the user's account i.e. creating and managing orders. All interfaces in this group are executed within the context of a user's session, and so the user's identity is inferred from that session and is not specified explicitly on each API call.

GetAccountBalances

Goal	Returns a summary of current balances.
Description	
Input Parameters	*None*
Output Parameters	*currency : Currency* *availableFunds : MoneyAmount* *balance : MoneyAmount* *credit : MoneyAmount* *exposure: MoneyAmount*
Return Codes	• RC406 PunterIsBlacklisted

ListAccountPostings

Goal

Returns more detailed information about the account, including account transactions between two given date and times.

Description

There is a limit on the number of records that will be returned by this API call. The records returned are guaranteed to be in order of increasing posting date and time. To obtain the full set of postings repeatedly call this API specifying the maximum postedAtTime returned from the previous call as the startTime until no more records are returned. (Note, Posting times are not absolutely unique and you must be able to cater for this when retrieving multiple pages of Postings.)

Note, there can be more than one Settlement posting for any order, and more than one Commission posting for any market - there will be more than one in the case of a market being settled, unsettled and then resettled.

Input Parameters

startTime : Timestamp
endTime : Timestamp

Output Parameters

resultSetTruncated : Boolean

Flag indicating whether or not all available data was returned. There is a system defined limit on the number of postings that will be returned in any one call (let's call it 'n'). If more than 'n' postings match the search criteria then only the forst 'n' will be returned and this flag will be true. If less than 'n' match the search criteria then all those postings will be returned and this flag will be false.

currency : Currency
availableFunds : MoneyAmount
balance : MoneyAmount
credit : MoneyAmount
exposure: MoneyAmount
[variable]

postedAt : Timestamp
description : String(256)
amount: MoneyAmount
transactionId : long
resultingBalance: MoneyAmount
postingCategory : PostingCategory
[optionally]

[either]

Present if postingCategory is Settlement.
handle : long <Order>

[or]

Present if postingCategory is Commission.
handle : long <Market>

Return Codes

- RC406 PunterIsBlacklisted

ListAccountPostingsById

Goal

Returns more detailed information about the account, including account transactions that have a transactionId greater than the value specified as input parameter.

Formatted

Description

This differs from ListAccountPostings in that this API returns all Postings that have a *transactionId* greater than the one specified as input parameter, whereas ListAccountPostings returns Postings that were created between specific times. A system-defined limit defines the maximum number of Postings that will be returned from any one call and if there are more Postings than that limit then only the first number of Postings will be returned and the caller will need to invoke this API again specifying as input parameter the maximum *transactionId* returned in the previous call. These Postings are sorted (ascending) by *transactionId*.

The anticipated usage is that the caller will first call

ListAccountPostings for the time period required, and then change to calling this API (specifying the maximum *transactionId* returned from the ListAccountPostings call as input parameter).

Note, there can be more than one Settlement posting for any order, and more than one Commission posting for any market - there will be more than one in the case of a market being settled, unsettled and then resettled.

Input Parameters

transactionId : long

Output Parameters

currency : Currency
availableFunds : MoneyAmount
balance : MoneyAmount
credit : MoneyAmount
exposure: MoneyAmount
[variable]
postedAt : Timestamp
description : String (256)
amount: MoneyAmount
transactionId : long
resultingBalance: MoneyAmount
postingCategory : PostingCategory
[optionally]
[either]
Present if postingCategory is Settlement.
orderId : long
[or]
Present if postingCategory is Commission.
marketId : long

Return Codes

- RC406 PunterIsBlacklisted

ListOrdersChangedSince

Goal

Returns a list of orders (for the currently logged in user) that have changed since a given sequence number.

Description

The number of records that are returned by this API call is limited to a system-defined value. It is guaranteed that those rows are returned in order of increasing sequenceNumber. Thus this API can be called repeatedly (specifying the maximum sequenceNumber received on the previous call) to obtain the full set.

This API call returns a record for every order that has had any change made since the sequenceNuumber specified. This includes any orders that have been settled or cancelled. This information may be useful if you want to track the settlement status of your orders.

As this API call returns information about all orders (including settled orders) it should not be used for initialisation. Rather the ListBootstrapOrders API should be used for initialisation, and when fully initialised you can switch to use this API.

Input Parameters

sequenceNumber : long

Output Parameters

[variable]
handle : long <Order>
handle : long <Market>
handle : long <Selection>
sequenceNumber : long
issuedAt : Timestamp
polarity : Polarity
unmatchedStake : MoneyAmount
totalForSideMakeStake : MoneyAmount
The amount of the matched for side stake that was a 'make'. The value returned here reflects only the amount of 'make' stake that was matched since the exchange started to record make/ take information and so the sum of

this and *totalForSideTakeStake* will not necessarily equal the *totalForSideStake* for orders that were created before that date.

totalForSideTakeStake : MoneyAmount
The amount of the matched for side stake that was a 'take'. The value returned here reflects only the amount of 'take' stake that was matched since the exchange started to record make/ take information and so the sum of this and *totalForSideMakeStake* will not necessarily equal the *totalForSideStake* for orders that were created before that date.

requestedPrice : Price
[optionally]
matchedPrice : Price
[optionally]
matchedStake : MoneyAmount
matchedAgainstStake : MoneyAmount
status : OrderStatus
orderFillType : OrderFillType
[optionally]
fillOrKillThreshold : MoneyAmount
expectedSelectionResetCount : int
expectedWithdrawalSequenceNumber : int
restrictOrderToBroker : Boolean
punterReferenceNumber : long
cancelOnInRunning : Boolean
cancelIfSelectionReset : Boolean
isCurrentlyInRunning : Boolean
punterCommissionBasis : PunterCommissionBasis
The basis on which commission is calculated.
makeCommissionRate : Percentage
The commission rate that was used in calculating commission for 'make' matches involving this order.
takeCommissionRate : Percentage
The commission rate that was used in calculating commission for 'take' matches involving this order.
[optionally]
Will only be present for Orders that have been settled.
grossSettlementAmount: MoneyAmount
The gross profit or loss for this order. Will only be present for settled orders.
[optionally]

orderCommission : MoneyAmount
The amount of commission that was charged on this Order. Will only be present if the Order was settled on any basis other than NetMarketWinnings (note that this refers to the basis on which the Order was settled and could be different to the basis associated with the Order when the Order was placed).

Return Codes

- RC406 PunterIsBlacklisted

ListBootstrapOrders

Goal

List bootstrap orders that have a sequence number greater than the sequence number specified.

Description

This call is used to obtain the initial list of orders that need to be taken into consideration when establishing positions. Information about the following orders will be returned:

- active orders
- fully matched orders
- cancelled orders that have a matched portion
- suspended orders
- some settled or voided orders under some conditions

    Orders that have been settled or voided are not usually returned by this API, however settled or voided orders on a market which has not been fully settled will be returned if a value of true is specified for the *wantSettledOrdersOnUnsettledMarkets* input parameters. (This situation can occur when there is a Selection that has been settled although other Selections in the Market have not yet been settled, for example teams that have been knocked out of the FA Cup can be settled as losing Selections before the winner of the FA Cup is known).

Cancelled orders that do not have a matched portion will not be returned by this call.

This call is to be used when obtaining the initial list of active orders. The ListOrdersChangedSince call should be used to poll for any changes to that position.

There is a limit on the number of records that this API returns on any one call. This limit (referred to as 'nnn' below) is in the hundreds. If there are more than 'nnn' records to be returned then only the first 'nnn' will be returned but it is guaranteed that they will be returned in the order of their punterSequenceNumber. Subsequent calls to this API can be used to return the rest of the records.

This API call returns a maximumPunterSequenceNumber. This is used to enable initial positions to be reliably established where there are more active orders than can be returned from one ListBootstrapOrders call. The way to use these is as follows. Initially call this API specifying -1 for punterSequenceNumber. In addition to returning up to 'nnn' records this API call also returns the current maximumPunterSequenceNumber. Now repeatedly call this API specifying the maximum punterSequenceNumber returned on the previous call until either:

1. you receive a punterSequenceNumber equal to or greater than the *maximumPunterSequenceNumber* returned from first call
2. or until no more records are returned.

At this stage you have fully bootstrapped and you can now change to polling using ListOrdersChangedSince (initially specifying *maximumPunterSequenceNumber* as the *punterSequenceNumber* input parameter to that call).

When you have completed the bootstrap cycle you are not guaranteed that you have a full set of orders. You are, however, guaranteed that when you subsequently use ListOrdersChangedSince that you will have that full set. For example, let's say you need to call this API 5 times to complete the bootstrap cycle. If after the first call an order that would have been returned in the third call gets modified. Its sequenceNumber is now greater than the sequence number at which you will terminate the bootstrap cycle. Therefore you will not have received that order when you complete the bootstrap cycle. However, you will receive that order when you call ListOrdersChangedSince subsequently.

Note that the meaning of maximumPunterSequenceNumber is that there are no bootstrap orders with a sequence number greater than the one specified, not that there is necessarily a bootstrap order with a sequence number equal to the maximumPunterSequenceNumber specified.

Input Parameters

sequenceNumber : long
[optionally]

wantSettledOrdersOnUnsettledMarkets : Boolean

Flag indicating whether or not information about settled orders on unsettled markets should be returned. Values are true: return information about settled orders on unsettled

markets and false: do not return information about settled orders on unsettled markets. If not specified a value of false is defaulted.

Information about settled orders in unsettled markets can be useful when calculating the punter's profit and loss in markets that have been partially settled.

Output Parameters

maximumSequenceNumber : long

This is the maximum punter sequence number for the punter when the call is issued. This can vary from call to call during the one bootstrap cycle. Use the value returned on the first call in the cycle for all remaining calls in the bootstrap cycle.

[variable]

handle : long <Order>
handle : long <Market>
handle : long <Selection>
sequenceNumber : long
issuedAt : Timestamp
polarity : Polarity
unmatchedStake : MoneyAmount
totalForSideMakeStake : MoneyAmount

The amount of the matched for side stake that was a 'make'.

totalForSideTakeStake : MoneyAmount

The amount of the matched for side stake that was a 'take'.

requestedPrice : Price
[optionally]
matchedPrice : Price
[optionally]
matchedStake : MoneyAmount
matchedAganistStake : MoneyAmount
status : OrderStatus
orderFillType : OrderFillType
[optionally]
fillOrKillThreshold : MoneyAmount
expectedSelectionResetCount : int
expectedWithdrawalSequenceNumber : int
restrictOrderToBroker : Boolean
punterReferenceNumber : long
cancelOnInRunning : Boolean
cancelIfSelectionReset : Boolean
isCurrentlyInRunning : Boolean
punterCommissionBasis : PunterCommissionBasis

The basis on which commission is calculated.

makeCommissionRate : Percentage

The commission rate that was used in calculating commission for 'make' matches involving this order.

takeCommissionRate : Percentage

The commission rate that was used in calculating commission for 'take' matches involving this order.

[optionally]
grossSettlementAmount: MoneyAmount

The gross profit or loss for this order. Will only be present for settled orders (which will only ever be returned if the *wantSettledOrdersOnUnsettledMarkets* input parameter was true).

Return Codes

- RC406 PunterIsBlacklisted

GetOrderDetails

Goal
Get detailed information about an order.

Description
This API returns full detail and history about an order. This API should not be called routingly but only in the exceptional case where there is some query or uncertainity about the status of a particular order.

Input Parameters
handle : long <order>

Output Parameters
handle : long <Selection>
status : OrderStatus
issuedAt : Timestamp
lastChangedAt : Timestamp
expiresAt : Timestamp
validFrom : Timestamp
restrictOrderToBroker : Boolean
orderFillType : OrderFillType
[optionally]
fillOrKillThreshold : MoneyAmount
handle : long <Market>
marketType: MarketType
status : MarketStatus
requestedStake : MoneyAmount
requestedPrice : Price
expectedSelectionResetCount : int
expectedWithdrawalSequenceNumber : int
totalStake : MoneyAmount
unmatchedStake : MoneyAmount
averagePrice : Price
matchingTimestamp : Timestamp
polarity : Polarity
withdrawRepriceOption : WithdrawRepriceOption
cancelOnInRunning : Boolean
cancelIfSelectionReset : Boolean
sequenceNumber : long
punterReferenceNumber : long
[optionally]
If the order has been settled then these values will be present.
grossSettlementAmount : MoneyAmount
This will be specified only if the status of the Order is Settled. The amount here is the gross amount posted as a result of this order being settled. A positive amount indicates a 'winning' bet and a negative amount indicates a 'loosing' bet.
[either]
orderCommission : MoneyAmount
The amount of commission that was charged on this Order.
[or]
marketCommission :MoneyAmount
The amount of commission that was charged on this market.
marketSettledDate : Timestamp
The time at which the market was settled.
[variable]
The following is an audit log of the entire history of the order. It is intended for human consumption only.
time : Timestamp
orderActionType : OrderActionType
[optionally]
requestedStake : MoneyAmount
The absolute amount specified (when placing

order) or the amount of change (when changing an order) – in which case the amount can benegative.

[optionally]
requestedPrice : Price
[optionally]
The amount matched in this match – will only be present when 'Matched'.
matchedStake : MoneyAmount
matchedAgainstStake : MoneyAmount
priceMatched : Price
handle : long <MatchedOrder>
[optionally]
This is optional to cater for Orders that were matched before this property was added. The property will be present for all Order matches that occurred after this property has been added.
wasMake : Boolean
Flag indicating whether or not the Order concerned was the 'make' or the 'take' order in this particular match. Values are true: the order concerned was the 'make' order in this match and false: the order concerned was the 'take' order in this match.

totalStake : MoneyAmount
The total amount of matched for side stake after the action referenced by this record occurred.
totalAgainstStake : MoneyAmount
The total amount of matched against side stake after the action referenced by this record occurred.
averagePrice : Price
[optionally]
grossSettlementAmount : MoneyAmount
orderCommission : MoneyAmount
The amount of commission that was charged on this Order.

Return Codes

- RC021 OrderDoesNotExist
- RC274 PunterOrderMismatch
- RC406 PunterIsBlacklisted

PlaceOrdersNoReceipt

Goal Description

Places one or more orders on the exchange.

This returns the handle of each order placed. It does not return any indication of the match status of the orders (and accordingly does not wait for the matching cycle to complete before returning).

Depending on the value of a parameter (*wantAllOrNothingBehaviour*) the placing of all orders specified will occur atomically – either all of them will be placed or none of them will be placed if an error occurs. Depending on the value of this parameter return code handling will be slightly different in this API than in other APIs. There are a number of return codes that could affect a specific order through no fault of the caller, for example if the remaining unmatched portion of an order has just got matched. Instead of rejecting the change to all orders a set of return codes has been defined that are considered acceptable return codes. If one of the acceptable return codes is encountered on one or more orders then the change to the orders concerned will not occur but the change to all other orders will occur. However if a return code outside of these acceptable

return codes occurs on any order then no orders will be changed by this API and that return code will be returned as the API return code. If the only return codes encountered on any order is an acceptable return code the API return code will be Success and the order-specified return code will contain the acceptable return code for the order(s) concerned. The acceptable return codes are:

- RC000 Success
- RC015 MarketNotActive
- RC017 SelectionNotActive
- RC114 ResetHasOccurred

Input Parameters

wantAllOrNothingBehaviour : Boolean

Flag controls whether all or nothing behaviour is desired. alues are *true*: do not place any oders if any error occurs and *false* if an acceptable error occurs placing an order proceed and place all other orders. If a non-acceptable error occurs placing any order then no orders will be placed.

<variable>

handle : long <Selection>
requestedStake : MoneyAmount
requestedPrice : Price
polarity : Polarity
expectedSelectionResetCount : short
expectedWithdrawalSequenceNumber : short
cancelOnInRunning : Boolean

A value of false must be specified when placing orders on an in-running market, otherwise RC114 ResetHasOccurred will be returned.

If it is desired to place an order in an non-in-running market and for it not to be cancelled when the market goes in running then a value of false must also be specified for cancelIfSelectionReset. If a value of false is specified for cancelOnInrunning and a value of true is specified for cancelIfSelectionReset then the order will be treated as if the value specified for cancelOnInrunning was true.

If a value of false is specified then the value actually specified for *withdrawRepriceOption* is ignored and a value of *DontReprice* is always used.

A value of *false* can be specified for back orders with an *OrderFillType* of *SPIfUnmatched.* In that case any part of the order that was not matched at SP for any reason when the market was turned in-running will remain available to be matched in the in-running market. It is not valid to specify a value of *false* for lay orders that have an *OrderFillType* of *SPIfUnmatched*. If a value of *false* is specified in this case the order will be treated as if a value of *true* was specified.

cancelIfSelectionReset : Boolean
[optionally]
expiresAt : Timestamp
withdrawRepriceOption : WithdrawRepriceOption
restrictOrderToBroker : Boolean

Deprecated.

[optionally]
killType : KillType

If not specified then the Order will have no kill type, that is, the order will never be cancelled automatically after the first attempt has been made to match it.

punterReferenceNumber : long

Does not need to be unique. Whatever value is specified here will be returned on all calls returning information about the order. Can be used by the caller to correlate the handle of an Order with an id of meaning to the caller.

[optionally]
channelInformation : String(256)

Output Parameters

<variable>
handle : long <Order>
returnCode : int
punterReferenceNumber : long

Return Codes

- RC011 SelectionDoesNotExist
- RC015 MarketNotActive
- RC017 SelectionNotActive
- RC022 NoUnmatchedAmount
- RC025 PunterReservationPerMarketExceeded
- RC114 ResetHasOccurred
- RC128 TradingCurrentlySuspended
- RC131 InvalidOdds
- RC136 WithdrawalSequenceNumberIsInvalid
- RC137 MaximumInputRecordsExceeded
- RC208 PunterSuspended
- RC240 PunterProhibitedFromPlacingOrders
- RC241 InsufficientPunterFunds
- RC271 OrderAPIInProgress
- RC302 PunterIsSuspendedFromTrading
- RC305 ExpiryTimeInThePast
- RC406 PunterIsBlacklisted
- RC597 MarketIsForRealMoney
- RC598 MarketIsForPlayMoney

PlaceOrdersWithReceipt

Goal Description

Places one or more orders on the exchange.

This call waits for a matching cycle to complete and returns information about how much of each order was matched and lists the status of each order. As it waits for a matching cycle to complete it takes more time than the PlaceOrdersNoReceipt call. This API can only be used for placing FillAndKill, FillOrKill or SPIfUnmatched orders.

Input Parameters

<variable>
handle : long <Selection>
requestedStake : MoneyAmount
requestedPrice : Price
polarity : Polarity
expectedSelectionResetCount : short
expectedWithdrawalSequenceNumber : short
killType : KillType
[optionally] <if FillOrKill or FillOrKillDontCancel >
fillOrKillThreshold : MoneyAmount
[optionally] <if FillOrKillDontCancel>
cancelOnInRunning : Boolean
A value of false must be specified when placing orders on an in-running market, otherwise RC114 ResetHasOccurred will be returned.
If it is desired to place an order in a pre-start market and for it not to be cancelled when the market goes in-running then a value of false must also be specified for cancelIfSelectionReset. If a

value of false is specified for cancelOnInrunning and a value of true is specified for cancelIfSelectionReset then the order will be treated as if the value specified for cancelOnInrunning was true.

If a value of false is specified then the value actually specified for *withdrawRepriceOption* is ignored and a value of *DontReprice* is always used.

It is not valid to specify a value of *false* for orders that have an *OrderFillType* of *SPIfUnmatched*. If a value of *false* is specified in this case the order will be treated as if a value of *true* was specified.

cancelIfSelectionReset : Boolean

It is not valid to specify a value of *false* for orders that have an *OrderFillType* of *SPIfUnmatched*. If a value of *false* is specified in this case the order will be treated as if a value of *true* was specified.

withdrawRepriceOption : WithdrawRepriceOption

[optionally]

expiresAt : Timestamp

restrictOrderToBroker : Boolean

Deprecated.

punterReferenceNumber : long

Does not need to be unique. Whatever value is specified here will be returned on all calls returning information about the order. Can be used by the caller to correlate the handle of an Order with an id of meaning to the caller.

[optionally]

channelInformation : String(256)

Output Parameters

<variable>

handle : long <Order>
sequenceNumber : long
issuedAt : Timestamp
polarity : Polarity
unmatchedStake : MoneyAmount
[optionally]
matchedPrice : Price
[optionally]
matchedStake : MoneyAmount
matchedAgainstStake : MoneyAmount
status:OrderStatus
punterReferenceNumber : long

Return Codes

- RC011 SelectionDoesNotExist
- RC015 MarketNotActive
- RC017 SelectionNotActive
- RC022 NoUnmatchedAmount
- RC114 ResetHasOccurred
- RC128 TradingCurrentlySuspended
- RC131 InvalidOdds
- RC136 WithdrawalSequenceNumberIsInvalid
- RC137 MaximumInputRecordsExceeded
- RC208 PunterSuspended
- RC240 PunterProhibitedFromPlacingOrders
- RC241 InsufficientPunterFunds
- RC271 OrderAPIInProgress

- RC302 PunterIsSuspendedFromTrading
- RC305 ExpiryTimeInThePast
- RC406 PunterIsBlacklisted
- RC597 MarketIsForRealMoney
- RC598 MarketIsForPlayMoney

UpdateOrdersNoReceipt

Goal
Updates one or more orders on the exchange.

Description
This changes the price or the amount (or both) of an existing orders. It does not return any indication of the match status of any of the orders concened (and accordingly does not wait for the matching cycle to complete before returning). It returns a separate return code for each order specified.
It is an error to attempt to change an order that is currently subject to an in-running delay.

Input Parameters
<variable>
handle : long <Order>
deltaStake : MoneyAmount
price : Price
expectedSelectionResetCount : short
expectedWithdrawalSequenceNumber : short
[optionally]
cancelOnInRunning : Boolean
If a value is specified for an order on an in-running market a value of false must be specified, otherwise RC114 ResetHasOccurred will be returned.
If a value of false is specified for cancelOnInrunning and a value of true is specified for cancelIfSelectionReset then the order will be treated as if the value specified for cancelOnInrunning was true.
[optionally]
cancelIfSelectionReset : Boolean
[optionally]
setToBeSPIfUnmatched : Boolean
Flag indicating that the *orderFillType* of this order should be changed to be *SPIfUnmatched*. This can only be specified if the *orderFillType* is currently *Normal* (if not, then RC892 CannotChangeToSPIfUnmatched will be returned). Values are: *true* change the *orderFillType* to *SPIfUnmatched* and *false* or not specified: do not change the *orderFillType*.

Output Parameters
[variable]
handle : long <Order>
returnCode : int

Return Codes
- RC015 MarketNotActive
- RC017 SelectionNotActive
- RC021 OrderDoesNotExist
- RC022 NoUnmatchedAmount
- RC114 ResetHasOccurred
- RC128 TradingCurrentlySuspended
- RC131 InvalidOdds
- RC136 WithdrawalSequenceNumberIsInvalid
- RC137 MaximumInputRecordsExceeded
- RC208 PunterSuspended
- RC240 PunterProhibitedFromPlacingOrders
- RC241 InsufficientPunterFunds
- RC271 OrderAPIInProgress

- RC274 PunterOrderMismatch
- RC293 InRunningDelayInEffect
- RC299 DuplicateOrderSpecified
- RC302 PunterIsSuspendedFromTrading
- RC305 ExpiryTimeInThePast
- RC306 NoChangeSpecified
- RC406 PunterIsBlacklisted
- RC892 CannotChangeToSPIfUnmatched

CancelOrders

Goal
Cancels one or more orders on the exchange.

Description
Information about each order specified is returned provided that there is no unmatched portion of that order existing at the time this API returns (that is, if the order was cancelled explicitly by this call, if it had been previously cancelled or if it had been fully matched). Orders that are currently subject to an in-running delay will not be cancelled immediately. Rather, when the in-running delay period has expired an attempt will be made to match the order and then any remaining unmatched amount on the order will be cancelled.

It is not considered an error if any of the orders specified can not be cancelled because they have been fully filled, already cancelled or are currently subject to an in-running delay. It is considered an error if the handle of an non-existent order was specified.

Information about every order specified will be returned regardless of whether or not the order was cancelled by this API except for orders that are currently subject to an in-running delay. Specifically, if an order specified was fully matched or had been cancelled by the system information about the current status of that order will be returned whereas if the order is currently subject to an in-running delay then no information about that order will be returned. Note that this can result in information about fewer orders being returned than was explicitly specified as input parameters,

Input Parameters
[variable]
handle : long <Order>

Output Parameters
[variable]
handle : long <Order>
punterReferenceNumber : long
cancelledForSideStake : MoneyAmount
The amout of for side stake that was cancelled. This amount is always definitive. (In some rare circumstances subsequent calls to e.g. ListOrderChangedSince may not immediately reflect amounts that were matched just prior to the order being cancelled. The amount of stake that was matched when the order was cancelled can calculated from this value, which is guaranteed to be correct).

Return Codes
- RC021 OrderDoesNotExist
- RC137 MaximumInputRecordsExceeded
- RC274 PunterOrderMismatch
- RC299 DuplicateOrderSpecified

CancelAllOrdersOnMarket

Goal
Cancels all unmatched orders on a market.

Description
Information about each order actually cancelled by this API will be returned. Information about orders currently subject to an in-running delay will not be returned as those orders will not be cancelled immediately.

Rather, when the in-running delay period has expired an attempt will be made to match the order and then any remaining unmatched amount on the order will be cancelled.

This API will explicitly determine the number of Orders to be actually cancelled and will not cancel any but return RC137 if the number of orders to be cancelled exceeds the limit. You must then find another mechanism to cancel those orders (like cancelling those orders in groups by specifying the orders handles expicitly). This is to protect against Denial of Service attacks.

Input Parameters

[variable]

handle : long <Market>

Output Parameters

[variable]

handle : long <Order>

punterReferenceNumber : long

cancelledForSideStake : MoneyAmount

The amout of for side stake that was cancelled. This amount is always definitive. (In some rare circumstances subsequent calls to e.g. ListOrderChangedSince may not immediately reflect amounts that were matched just prior to the order being cancelled. The amount of stake that was matched when the order was cancelled can calculated from this value, which is guaranteed to be correct).

Return Codes

- RC008 MarketDoesNotExist
- RC016 MarketNeitherSuspendedNorActive
- RC137 MaximumInputRecordsExceeded

CancelAllOrders

Goal Description

Cancels all unmatched orders across all markets.

Information about each order actually cancelled by this API will be returned. Information about orders currently subject to an in-running delay will not be returned as those orders will not be cancelled immediately. Rather, when the in-running delay period has expired an attempt will be made to match the order and then any remaining unmatched amount on the order will be cancelled.

This API will explicitly determine the number of Orders to be actually cancelled and will not cancel any but return RC137 if the number of orders to be cancelled exceeds the limit. You must then find another mechanism to cancel those orders (like cancelling those orders in groups by specifying the orders handles expicitly). This is to protect against Denial of Service attacks.

Input Parameters

Output Parameters

[variable]

handle : long <Order>

punterReferenceNumber : long

cancelledForSideStake : MoneyAmount

The amout of for side stake that was cancelled. This amount is always definitive. (In some rare circumstances subsequent calls to e.g. ListOrderChangedSince may not immediately reflect amounts that were matched just prior to the order being cancelled. The amount of stake that was matched when the order was cancelled can calculated from this value, which is guaranteed to be correct).

Return Codes

- RC137 MaximumInputRecordsExceeded

ListBlacklistInformation

Goal Description

Lists the black-list status for the punter.

This returns a list of every API from which the Punter is currentltly black-

listed along with the remaining time (in milli-seconds) of the black-list period for that API. Punters can be black-listed for different periods for difference APIs.

Input Parameters

Output Parameters *[variable]*

apiName : String(32)

remainingMS : int

Number of milli-seconds remaining until the black-list period expires.

SuspendFromTrading

Goal Suspend any of your orders from being matched.

Description This instantly prohibits any of your orders from getting matched subsequently. It is very quick and efficient to suspend yourself in this way, but it can take considerable effort to unsuspend yourself subsequently (all your open orders must be suspended or cancelled before you can be unsuspended).

It is intended that this API be used sparingly and only in emergencies.

Input Parameters

Output Parameters

Return Codes

- RC302 PunterIsSuspendedFromTrading
- RC406 PunterIsBlacklisted

UnsuspendFromTrading

Goal Unsuspend yourself from being suspending from trading.

Description This will allow your orders to be subsequently matched.

You may not have any active orders when the attempt is made to Unsuspend yourself – all active orders at the you were suspended must be cancelled or suspended before you can be unsuspended.

Input Parameters

Output Parameters

Return Codes

- RC303 PunterHasActiveOrders
- RC304 PunterNotSuspendedFromTrading
- RC406 PunterIsBlacklisted

SuspendOrders

Goal Suspends one or more orders.

Description The amount of funds to be reserved is not re-calculated as a result of suspending orders.

Input Parameters *[variable]*

handle : long <Order>

Output Parameters *[variable]*

handle : long <Order>

punterReferenceNumber : long

suspendedForSideStake : MoneyAmount

The amout of for side stake remaining unmatched when the order was suspended. This amount is always definitive. (In some rare circumstances subsequent calls to e.g. ListOrderChangedSince may not immediately reflect amounts that were matched just prior to the order being suspended. The amount of stake that was matched when the order was suspended can calculated from this value,

which is guaranteed to be correct).

Return Codes

- RC021 OrderDoesNotExist
- RC137 MaximumInputRecordsExceeded
- RC274 PunterOrderMismatch
- RC299 DuplicateOrderSpecified

SuspendAllOrdersOnMarket

Goal
Suspends all unmatched orders on a market.

Description
The amount of funds to be reserved is not re-calculated as a result of suspending orders.

This API will explicitly determine the number of Orders to be actually suspended and will not suspend any but return RC137 if the number of orders to be suspended exceeds the limit. You must then find another mechanism to suspend those orders (like suspending those orders in groups by specifying the orders handles expicitly). This is to protect against Denial of Service attacks.

Input Parameters
[variable]
handle : long <Market>

Output Parameters
[variable]
handle : long <Order>
punterReferenceNumber : long
suspendedForSideStake : MoneyAmount
The amout of for side stake remaining unmatched when the order was suspended. This amount is always definitive. (In some rare circumstances subsequent calls to e.g. ListOrderChangedSince may not immediately reflect amounts that were matched just prior to the order being suspended. The amount of stake that was matched when the order was suspended can calculated from this value, which is guaranteed to be correct).

Return Codes

- RC008 MarketDoesNotExist
- RC016 MarketNeitherSuspendedNorActive
- RC137 MaximumInputRecordsExceeded

SuspendAllOrders

Goal
Suspends all unmatched orders across all markets.

Description
The amount of funds to be reserved is not re-calculated as a result of suspending orders.

This API will explicitly determine the number of Orders to be actually suspended and will not suspend any but return RC137 if the number of orders to be suspended exceeds the limit. You must then find another mechanism to suspend those orders (like suspending those orders in groups by specifying the orders handles expicitly). This is to protect against Denial of Service attacks.

Input Parameters

Output Parameters
[variable]
handle : long <Order>
punterReferenceNumber : long
suspendedForSideStake : MoneyAmount
The amout of for side stake remaining unmatched when the order was suspended. This amount is always definitive. (In some rare circumstances subsequent calls to e.g. ListOrderChangedSince may not immediately reflect amounts that were matched just prior to the order being suspended. The amount of stake that was matched when the order was suspended can calculated from this value, which is guaranteed to be correct).

Return Codes

- RC137 MaximumInputRecordsExceeded

UnsuspendOrders

Goal Description Unsuspends one or more suspended orders.

Input Parameters *[variable]*
handle : long <Order>

Output Parameters

Return Codes

- RC021 OrderDoesNotExist
- RC137 MaximumInputRecordsExceeded
- RC274 PunterOrderMismatch
- RC299 DuplicateOrderSpecified
- RC302 PunterIsSuspendedFromTrading
- RC406 PunterIsBlacklisted

RegisterHeartbeat

Goal Description Register the Punter as requiring a Heartbeat.

Heartbeat provides a mechanism through which all of a Punter's orders will be cancelled, suspended (or the Punter itself suspended) automatically if connectivity is lost between the application and the system. This provides added protection to an application as it can be assured that its open orders will not get matched if it can no longer manage its position because it has lost connectivity with the system.

The basic mechanism is that an application specifies that it wants to create a Heartbeat and it specifies (i) the maximum threshold time after which an action is to be automatically performed if a Pulse is not received and (ii) the action that is to be performed (cancel all orders, suspend all orders or suspend the Punter). The application then sends Pulses (via the Pulse API) at least as frequently as the threshold specified. If a period of time greater than the threshold passes without the system having received a Pulse then the system automatically takes the action concerned. This action will only be performed once for each threshold period exceeded. For example, if the threshold value is 6000 and a period of 20000 milli-seconds occurs between Pulses being received. The appropriate action will be taken by the system 6000 milli-seconds after the Pulse was received. No further action will be taken until a subsequent Pulse API call is received, after which the system will again take the appropriate action if a Pulse API is not received the threshold time.

Heartbeat registrations are not persisted by the system. In the unlikely event that a system component fails it discards all heartbeat registrations when re-started, and all applications must reregister their Heartbeats. The application is notified of this situation via the RC462 PunterNotRegisteredForHeartbeat return code from the Pulse API. This could lead to exposure if both connectivity is lost and the system component is restarted at exactly the same time, but this is an extremely remote possibility.

Note that this mechanism acts at the Punter level. Only one registration can be active for a Punter at one time, and if a pulse is not received within that value the action specified will be performed. Although only one Heartbeat registration can be active at any time Pulse APIs can be issued from any source. Care must be taken if two applications are operating on the same Punter. At least two situations could arise:

1. If connectivity from the application that is issuing the Pulse APIs is lost then the action automatically performed will affect orders issued from all other applications (or issued manually through the

web site).

2. If two applications are issuing Pulse APIs for the one Punter and one of them lose connectivity the Pulse APIs being received from the other will have the effect that no action is taken for that Punter (as the system will keep received Pulse APIs for the relevant Punter).

Input Parameters

thresholdMs : int

The maximum period (in milli-seconds) that can elapse between Pulse API calls being received before the system takes the relevant action. A value of 6000 (equivelant to 6 seconds) is suggested, and is is suggested that the application issue Pulse API calls every 5 seconds.

An minimum threshold value may be enforced by the implementation for efficiency reasons. If a threshold value less than this minimum is specified it will be ignored and the minimum value used as the threshold. It is unlikely that this minimum threshold value will be less than 1000 milliseconds.

heartbeatAction : HeartbeatAction

The action that should be taken if a Pulse is not received within the threshold. If CancelOrders or SuspendOrders is specified and if the number of unmatched orders exceeds a limit the action that will actually be taken is that the Punter will be suspended. However the Punter would have been suspended more quickly had the action been specified as SuspendPunter in the first case.

Output Parameters

Return Codes

- RC463 PunterAlreadyRegisteredForHeartbeat

ChangeHeartbeatRegistration

Goal

Update the Heartbeat parameters for the Punter.

Description

Input Parameters

thresholdMs : int
heartbeatAction : HeartbeatAction

Output Parameters

Return Codes

- RC462 PunterNotRegisteredForHeartbeat

DeregisterHeartbeat

Goal

Deregister the Punter from requiring a Heartbeat.

Description

The effect of this is that no action will be automatically taken on the Punter if a Pulse is not received within the applicable threshold.

Input Parameters

Output Parameters

Return Codes

- RC462 PunterNotRegisteredForHeartbeat

Pulse

Goal

Notify the system that the application is still active and still has connectivity.

Description

If a period of time greater than the threshold (specified on a previous RegisterHeartbeat or ChangeHeartbeatRegistration API) has elapsed since the last Pulse was received the relevant action will be automatically performed by the system (e.g. all orders will be cancelled or suspended or the punter will be suspended automatically). The effect of this Pulse API is that the absolute time by which the next Pulse API must be received will be extended by the threshold value.

Heartbeat registrations are not persisted by the system. In the unlikely event that a system component fails it discards all heartbeat

registrations when re-started, and all applications must reregister their Heartbeats. The application is notified of this situation via the RC462 PunterNotRegisteredForHeartbeat return code from the Pulse API. This could lead to exposure if both connectivity is lost and the system component is restarted at exactly the same time, but this is an extremely remote possibility.

Input Parameters

Output Parameters

[optionally]

If an action was automatically performed since the last RegisterHeartbeat API call was issued then information about that action is returned.

performedAt : Timestamp

The time at which the action was performed.

heartbeatAction : HeartbeatAction

The actual action that was performed, which may differ from the action requested on RegisterHeartbeat. For example, the action requested may have been CancelOrders. However if the number of unmatched orders exceed a certain limit the action action that was performed would be SuspendPunter.

Return Codes

- RC462 PunterNotRegisteredForHeartbeat

ReadOnly

This group of interfaces deals with all aspects of read-only interaction with the API. As a rule the ReadOnly interface contain all methods that are not specific to the user's account i.e. getting Events, Markets, Selections and Prices etc.

ListTopLevelEvents

Goal Description
Returns the set of top level events that are currently active.

Input Parameters
language : String(2)
[optionally]
wantPlayMarkets : Boolean
Flag indicating whether information about play markets or real markets should be returned. Values are true: only information about play markets is to be returned and false: only information about real markets should be returned. If not specified a value of false is assumed.

Output Parameters
[variable]
handle : long <Event>
name : String
displayOrder : short
isEnabledForMultiples : Boolean

Return Codes
- RC406 PunterIsBlacklisted

GetEventSubTreeNoSelections

Goal Description
Returns the tree of events and markets.

Returns information about all descendent Events and Markets (but does not return Selection information).
Only information about active and suspended markets are returned. Events that do not directly or indirectly contain active or suspended markets are not returned.

Input Parameters
language : String(2)
[variable]
handle : long <Event>
wantDirectDescendentsOnly : Boolean
Values are true: return only direct descendents of the Events specified and false: return direct and indirect descendents of the Events specified.
[optionally]
wantPlayMarkets : Boolean
Flag indicating whether information about play markets or real markets should be returned. Values are true: only information about play markets is to be returned and false: only information about real markets should be returned. If not specified a value of false is assumed.

Output Parameters
[variable]
handle : long <Event>
parentHandle : long <Event>
name : String
displayOrder : short
isEnabledForMultiples : Boolean
[variable]
name : String
handle : long <Market>

parentHandle : long <Event>
type : MarketType
isPlayMarket : Boolean
status : MarketStatus
numberOfWinningSelections : short
startTime : Timestamp
withdrawalSequenceNumber : short
displayOrder : short
isEnabledForMultiples : Boolean
isInRunningAllowed : Boolean
[optionally]
raceGrade : String (2048)
String containing the encoded race grade and prize money information, if available.
managedWhenInRunning : Boolean
isCurrentlyInRunning : Boolean
inRunningDelaySeconds : int
[optionally]
placePayout : Percentage
The proportion of the payout that is to be paid on the place part of an order on an each-way market. Will only be present if the MarketType is one of *EachWayNonHandicap*, *EachWayHandicap* or *EachWayTournament.*

Return Codes

- RC005 EventClassifierDoesNotExist
- RC137 MaximumInputRecordsExceeded
- RC406 PunterIsBlacklisted

GetEventSubTreeWithSelections

Goal
Returns the tree of events and markets.

Description
Returns information about all descendent Events and Markets (including Selection information).

Only information about active and suspended markets are returned. Events that do not directly or indirectly contain active or suspended markets are not returned.

Information is returned about all selections within active and suspended markets regardless of the status of those selections.

Input Parameters
language : String(2)
[variable]
handle : long <Event>
[optionally]
wantPlayMarkets : Boolean
Flag indicating whether information about play markets or real markets should be returned. Values are true: only information about play markets is to be returned and false: only information about real markets should be returned. If not specified a value of false is assumed

Output Parameters
[variable]
handle : long <Event>
parentHandle : long <Event>
name : String
displayOrder : short
isEnabledForMultiples : Boolean
[variable]
name:String
handle : long <Market>
type : MarketType
isPlayMarket : Boolean
status : MarketStatus

numberOfWinningSelections : short
startTime : Timestamp
withdrawalSequenceNumber : short
dislpayOrder : short
isEnabledForMultiples : Boolean
isInRunningAllowed : Boolean
[optionally]
raceGrade : String (2048)
String containing the encoded race grade and prize money information, if available.
managedWhenInRunning : Boolean
isCurrentlyInRunning : Boolean
inRunningDelaySeconds : int
[optionally]
placePayout : Percentage
The proportion of the payout that is to be paid on the place part of an order on an each-way market. Will only be present if the MarketType is one of *EachWayNonHandicap*, *EachWayHandicap* or *EachWayTournament.*
[variable]
name : String
handle : long <Selection>
status : SelectionStatus
selectionResetCount : short
selectionDeductionFactor : Decimal
displayOrder : short

Return Codes

- RC005 EventClassifierDoesNotExist
- RC137 MaximumInputRecordsExceeded
- RC406 PunterIsBlacklisted

GetMarketInformation

Goal Description

Returns detailed information about a given market.

Input Parameters

language : String(2)
[variable]
handle : long <Market>

Output Parameters

[variable]
name : String
handle : long <Market>
parentHandle : long <Event>
type : MarketType
isPlayMarket : Boolean
status : MarketStatus
numberOfWinningSelections : short
startTime : Timestamp
withdrawalSequenceNumber : short
isEnabledForMultiples : Boolean
isInRunningAllowed : Boolean
[optionally]
raceGrade : String (2048)
String containing the encoded race grade and prize money information, if available.
managedWhenInRunning : Boolean
isCurrentlyInRunning : Boolean
inRunningDelaySeconds : int
[optionally]
placePayout : Percentage

The proportion of the payout that is to be paid on the place part of an order on an each-way market. Will only be present if the MarketType is one of *EachWayNonHandicap*, *EachWayHandicap* or *EachWayTournament*.

[variable]
name : String
handle: long <Selection>
status : SelectionStatus
selectionResetCount : short
selectionDeductionFactor : Decimal
displayOrder : short

Return Codes

- RC008 MarketDoesNotExist
- RC137 MaximumInputRecordsExceeded
- RC406 PunterIsBlacklisted

ListSelectionsChangedSince

Goal

Poll to see if any selections have changed since the previous poll.

Description

This API provides an efficient mechanism for external applications to determine when any selection has changed on GBE. Although this will inform of any changes the primary motivation of this API is to enable external applications to determine when Selections have been settled on the exchange.

A selectionSequenceNumber is specified as input parameter to this API. Details about any selections that have a selectionSequenceNumber greater than that sequenceNumber (i.e. that have been changed since that sequence number was allocated) will be returned. The returned details includes the new sequence number for each selection. The maximum sequence number returned from one call should be used as input to the next ListSelectionChangedSince call.

Note that the implementation will limit the number of selections that will be returned in any one call (probably to 500). However subsequence calls can be used to get the full set of selections that have been changed. It is recommended that on start-up that this API be called repeatedly until there are no new selections returned (at which stage the caller can be sure that there are no outstanding selection changes) and then called on a timed basis.

Input Parameters

language : String(2)
selectionSequenceNumber : long

Output Parameters

[variable]
name : String (256)
handle : long
displayOrder: int
isHidden : Boolean
status : SelectionStatus
selectionResetCount : int
withdrawalFactor : Percentage
[optionally]
cancelOrdersTime : Timestamp
[optionally]
settledTime : Timestamp
voidPercentage : Percentage
leftSideFactor : Percentage
rightSideFactor : Percentage
resultString : String (256)
handle : long <Market>
selectionSequenceNumber : long

Return Codes

-

ListMarketWithdrawalHistory

Goal Description Get history of withdrawals from this market.

Input Parameters *handle : long <Market>*

Output Parameters *[variable]*

handle : long <Selection>

withdrawalTime : Timestamp

sequenceNumber : int

Any orders with a withdrawal sequence number less than this need to be re-priced.

reductionFactor : Percentage

The deduction for this on specific withdrawal.

compoundReductionFactor : Percentage

The combined deduction to be applied to orders that have a withdrawal sequence number of the value specified.

Return Codes

- RC008 MarketDoesNotExist

GetPrices

Goal Description Returns the prices for a particular market.

Prices are sorted in order of decreasing competiveness (e.g. highest back price first, lowest lay price first).

In general, if an error occurs with one market then the prices for no markets are returned. For example, if 4 market handles are specified and one of them doesn't exist then no prices will be returned and a return code of of the handle of a market that doesn't exist is specified then a RC008 MarketDoesNotExist will be returned. However, and exception is made in the case that is not currently active or suspended (as a market could have just been completed without the caller being able to know this). In this case prices for other markets will be returned and in indication that the specific market is currently neither active nor suspended will also be returned.

Input Parameters

currency : String(3)

thresholdAmount : MoneyAmount

The minimum backers stake required for for a price. This is a mechanism to be able to set a threshold such that offers less than the threshold will not returned. For example, if you are only interested in a price if there is at least 100.00 available at that price then specify 100.00 for this value. Any prices that have less than 100.00 available will not be returned to you. Specify 0 if you do not have any threshold amount.

numberForPricesRequired : int

The maximum number of for prices to return for each selection. This is a mechanism to reduce the amount of data returned if you are only interested in, for example, the top 1 or top 3 prices on the for side. Specify -1 if you want all for prices returned or 0 if you do not want any for prices returned.

numberAgainstPricesRequired : int

The maximum number of against prices to return for each selection. This is a mechanism to reduce the amount of data returned if you are only interested in, for example, the top 1 or top 3 prices on the against side. Specify -1 if you want all against prices returned or 0 if you do not want any against prices returned.

[variable]

handle : long <Market>

[optionally]

wantMarketMatchedAmount : Boolean

Flag indicating whether or not the total amount matched on the market should be returned (specifically whether the *matchedMarketForStake* and *matchedMarketAgainstStake* output parameters should be returned). Values are *true*: return market matched information and *false*: don't return market matched information. If no value is specified then a value of false is assumed.

[optionally]

wantSelectionsMatchedAmounts : Boolean

Flag indicating whether or not the total amount matched on each selection should be returned

(specifically whether the *matchedSelectionForStake* and *matchedSelectionAgainstStake* output parameters should be returned). Values are *true*: return selection matched information and *false*: don't return selection matched information. If no value is specified then a value of false is assumed.

[optionally]

wantSelectionMatchedDetails : Boolean

Flag indicating whether or not details about the last match occurring on each selection should be returned (specifically whether the *lastMatchedOccurredAt, lastMatchedPrice, lastMatchedForSideAmount* and *lastMatchWasFor* output parameters should be returned). Values are *true*: return details about the last match occurring on each selection *false*: don't return those details. If no value is specified then a value of false is assumed.

Output Parameters

[variable]

[either]

handle : long <Market>
type : MarketType
isPlayMarket : Boolean
status : MarketStatus
numberOfWinningSelections : short
startTime : Timestamp
withdrawalSequenceNumber : short
isInRunningAllowed : Boolean
managedWhenInRunning : Boolean
isCurrentlyInRunning : Boolean
inRunningDelaySeconds : int
[optionally]
homeTeamScore : int
The score for the home team.
awayTeamScore : int
The score for the away team.
scoreType : ScoreType
[optionally]
totalMatchedAmount : MoneyAmount
The total amount that has been matched on the market. This will be returned for some markets and not for others.
[optionally]
placePayout : Percentage
The proportion of the payout that is to be paid on the place part of an order on an each-way market. Will only be present if the MarketType is one of *EachWayNonHandicap, EachWayHandicap* or *EachWayTournament.*
[optionally]
Only present if the value of the *wantMarketMatchedAmount* input parameter was true.
matchedMarketForStake : MoneyAmount
The total amount of for side stake matched on the market. For example, if only a single match has occurred on the market where that match was for $10 at a price of 11.0 then the value returned would be $10.
matchedMarketAgainstStake : MoneyAmount
The total amount of against side stake matched on the market. For example, if only a single match has occurred on the market where that match was for $10 at a price of 11.0 then the value returned would be $100.
[variable]
handle: long <Selection>
status : SelectionStatus
selectionResetCount : short
selectionDeductionFactor : Decimal
[variable]<ForSidePrices>

price : Price
stake : MoneyAmount

[variable]<AgainstSidePrices>
price : Price
stake : MoneyAmount

[optionally]
Only present if the value of the *wantSelectionsMatchedAmounts* input parameter was true.

matchedSelectionForStake : MoneyAmount
The total amount of for side stake matched on the selection. For example, if only a single match has occurred on the selection where that match was for $10 at a price of 11.0 then the value returned would be $10.

matchedSelectionAgainstStake : MoneyAmount
The total amount of against side stake matched on the selection. For example, if only a single match has occurred on the selection where that match was for $10 at a price of 11.0 then the value returned would be $100.

selectionOpenInterest : MoneyAmount
The current open interest in this Selection independently of all other Selections in this Market. This is the sum of the absolute value of the open selection interest of each punter who has matched orders on this Selection. The open selection interest for each punter is the difference between what the punter would win or lose on orders on this Selection if the Selection wins and if the Selection loses.

For example, consider the case where there are only two punters with matched orders on the Selection. If the Selection wins punter A will have a profit of $10 and punter B will have a loss of $10 while if the Selection loses punter A will have a loss of $5 and punter B will have a profit of $5. This value would be $30 (the difference between what punter A would have depending on whether the selection wins or loses is $15 and similarly for punter B).

[optionally]
These will only be present for markets that have a value of 1 for *numberOfWinningSelections. marketWinnings : MoneyAmount*
The current market winnings that would arise on this Selection taking into consideration the orders all punters have on other Selections in the Market. This is the sum of the absolute value of the market winnings for each punter who has orders matched in the Market (including those punters who do not actually have any orders matched on this Selection). The market winnings is the amount of money a punter would win or lose on all orders the punter has in the Market depending on whether the Selection wins or loses.

For example, consider a punter who has a single back order for $10 at 11.0 on Selection 1 and a single lay order for $20 at 3.0 on Selection 5. That punter's *selectionOpenInterest* on Selection 1 would be $110, his *marketOpenInterest* on Selection 1 would be $120 on Selection 1, -$50 on Selection 5 and $10 on all other selections.

marketPositiveWinnings : MoneyAmount
The sum of the positive market winnings that punters would have if this Selection wins.

[optionally]
Only present if (i) the value of the *wantSelectionMatchedDetails* input

parameter was true and (ii) at least one match has occurred on this selection.

lastMatchedOccurredAt : Timestamp
The time at which the last match occurred on this selection.

lastMatchedPrice : Price
The price at which the last matched occurred on this selection.

lastMatchedForSideAmount : MoneyAmount
The for side stake of the last match that occurred on this selection.

lastMatchedAgainstSideAmount : MoneyAmount
The for side stake of the last match that occurred on this selection.

matchedForSideAmountAtSamePrice : MoneyAmount
The total amount of for side stake that was matched at the same contiguous price as the last match that occurred on this Selection. For example, if $10 was matched at 1.98, then $8 at 2.0, then $5 at 1.98 and then $20 at 1.98 this value of this would be $25.

matchedAgainstSideAmountAtSamePrice : MoneyAmount
The total amount of against side stake that was matched at the same contiguous price as the last match that occurred on this Selection.

firstMatchAtSamePriceOccurredAt : Timestamp
The time (UTC) at which the first match occurred at the same contiguous price as the last match that occurred on this Selection – in other words the time since which all matching on the Selection has occurred at the same price.

numberOrders : int
The number of orders that have been matched on this Selection.

numberPunters : int
The number of punters who have had orders matched on this Selection.

[or]

returnCode : int
The only return code that is individually returned is RC016 MarketNeitherSuspendedNorActive.

Return Codes

- RC008 MarketDoesNotExist
- RC016 MarketNeitherSuspendedNorActive
- RC137 MaximumInputRecordsExceeded
- RC406 PunterIsBlacklisted

GetOddsLadder

Goal Description

Obtain the current odds ladder.

It is recommended that clients dynamically obtain the odds ladder (as opposed to hard-coding it) as the odds ladder may periodically change, which could cause clients that have hard-coded the odds ladder to stop working correctly.

The odds ladder returned identifies the only prices at which orders should be placed. It does not necessarily mean that prices returned (from GetPrices) will only contain those prices (it is possible that for short periods after a change in the odds ladder additional prices may be returned).

Input Parameters

priceFormat : PriceFormat

Output Parameters

[variable]

price : Price
The precise (to 12 places of decimal) price concerned.

representation : String
The way in which the price concerned should be displayed in the PriceFormat concerned. Consider the price of 3.00. The representation in Decimal would be '3', in Fractional it would be '2/1' and in American it would be '200'.

GetSPEnabledMarketsInformation

Goal Get information defining which markets are enabled for starting-price orders.

Description A list of *eventIds* is returned and for each *eventId* a list of *marketTypes* is also returned. A market is enabled for starting-price orders if both:

1. an *eventId* is listed that is a direct or indirect ancestor of the market concerned and
2. the *marketType* listed for that event is the same as that of the market concerned.

Input Parameters

Output Parameters

[variable]
eventId : long
[variable]
marketType: MarketType

Return Codes

-

GetCurrentSelectionSequenceNumber

Goal Get the current maximum selectionSequenceNumber.

Description This API provides the initial sequence number for use by ListSelectionChangedSince.

Input Parameters

Output Parameters *selectionSequenceNumber : long*

Return Codes

-

ListSelectionTrades

Goal Returns the history of trades on the selection(s) specified.

Description There is a limit on the number of trades about which this call returns information and this API may need to be called a number of times to get a full history of trades on the selections concerned. Furthermore, it can be polled to obtain information about any trades that occurred since its last invocation.

Each trade has an associated *tradeId* and *tradeIds* increase with time, although not monotonically. This call returns the maximum *tradeId* of trades about which information is returned (*maxTradeIdReturned*) and it also returns the *tradeId* of the last trade that occurred on the selection concerned (*maxTradeId*). An optional input parameter specifies the minimum *tradeId* about which information should be returned (*fromTradeId*). By repeatedly calling this API and specifying the *maxTradeIdReturned* returned from the previous call as the *fromTradeId* input parameter the full set of trades that occurred on the selection concerned can be obtained.

Input Parameters *currency : String (3)*
[variable]
selectionId : long
It is not considered to be an error condition if a non-existent or non-active *selectionId* is specified – simply no no trade information will be returned for that selection.
[optionally]
fromTradeId : long
If specified then only information about trades that have a *tradeId* greater than this value will be returned (trades are returned in order of increasing *tradeId*).

Output *[variable]*
selectionId : long

Formatted

Formatted

Parameters

maxTradeIdReturned : long
The maximum *tradeId* of the trades returned by this invocation (trades are returned in order of increasing *tradeId*).
maxTradeId : long
The *tradeId* of the latest trade that occurred on the selection concerned.
[variable]
occurredAt : Timestamp
price : Price
backersStake : MoneyAmount
The backer's stake of the trade concerned in the currency concerned.
layersLiability : MoneyAmount
The layer's liability of the trade concerned in the currency concerned.
tradeType : TradeType
An indication of whether this was a back or a lay.

Return Codes

- RC137 MaximumInputRecordsExceeded
- RC406 PunterIsBlacklisted

ListTaggedValues

Goal Return Exchange TaggedValues.

Description Returns tagged values for the entities (EventClassifier, Market or Selection) concerned

Input Parameters

[variable]
identifier : long
entityType : EntityType
[optionally]
[variable]
name : String (256)
Cannot contain wildcards.
[optionally]
wantDescendents : Boolean
This parameter is ignored if the entity type is not EventClassifier or Market. If the entityType is EventClassifier or Market and if a value for this (wantDescendents) parameter is not specified then a value of false is defaulted.
This parameter controls whether TaggedValues for only the EventClassifier or Market specified or for the EventClassifier or Market specified (and all Markets directly contained within that EventClassifier) and all contained Selections. Values are true: return tagged values for the EventClassifier specified (and for Markets directly contained within it) and for contained Selections contained and false: only return TaggedValues explicitly defined for the EventClassifier or Market specified.

Output Parameters

[variable]
identifier : long
[variable]
entityType : entityType
name : String (256)
value : String (unlimited)

Return Codes

- RC137 MaximumInputRecordsExceeded
- RC406 PunterIsBlacklisted

Formatted

Data Dictionary (enumerations)

Boolean

Currency Character string containing the 3 character ISO currency code.

Decimal

EntityType Enumeration defining type of object. Domain values are:
EventClassifier (1)
Market (2)
Selection (3)
AuthorisedUser (14)

HeartbeatAction Enumeration defining the action to be performed when a threshold period has expired without a Pulse having been received. Domain values are:

CancelOrders (1)
Explicitly cancel all unmatched orders.

SuspendOrders (2)
Explicitly suspend all unmatched orders.

SuspendPunter (3)
Explicitly suspend the punter.

KillType Enumeration defining the kill type of a order. Domain values are:

FillAndKill (2)
After the initial attempt is made to match this order any unmatched portion of the order is immediately cancelled. Thus the order can be partially matched but there will never be any unmatched portion of the order remaining.

FillOrKill (3)
On the initial attempt to match this order if it is not possible to match a specified amount of the order then none of the order will be matched. If it had been possible to match at least the specified amount then the amount that can be matched will be matched and the remaining unmatched amount will be cancelled.

FillOrKillDontCancel (4)
On the initial attempt to match this order if it is not possible to match a specified amount of the order then none of the order will be matched. If it had been possible to match at least the specified amount then the amount that can be matched will be matched and the remaining unmatched amount will be not be cancelled but left as an unmatched order.

SPIfUnmatched (5)
Same as *Normal* but any unmatched portion of the order is to be matched at SP when the market is turned in-running (or completed).

long

MarketStatus Enumeration defining the current status of a market. Domain values are:

Inactive (1)
The Market is not active and has never had any Orders issued against it.

Active (2)
The Market it active (that is, Orders can be issued

Formatted

against it).

Suspended (3)

The Market is not currently active but it has not yet been completed.

Completed (4)

The Market is completed. No further Orders can be issued against the Market but the result of the Market is either not yet known or has not yet been entered.

Settled (6)

The Market has been fully settled.

Voided (7)

The Market has been voided. All matched Orders in this Market have also been voided.

MarketType Enumeration defining the type of a market. Domain values include:

Win (1)
Place (2)
MatchOdds (3)
OverUnder (4)
AsianHandicap (10)
TwoBall (11)
ThreeBall (12)
Unspecified (13)
MatchMarket (14)
SetMarket (15)
Moneyline (16)
Total (17)
Handicap (18)
EachWayNonHandicap (19)
EachWayHandicap (20)
EachWayTournament (21)
RunningBall (22)
MatchBetting (23)
MatchBettingInclDraw (24)
CorrectScore (25)
HalfTimeFullTime (26)
TotalGoals (27)
GoalsScored (28)
Corners (29)
OddsOrEvens (30)
HalfTimeResult (31)
HalfTimeScore (32)
MatchOddsExtraTime (33)
CorrectScoreExtraTime (34)
OverUnderExtraTime (35)
ToQualify (36)
DrawNoBet (37)
HalftimeAsianHcp (39)
HalftimeOverUnder (40)
NextGoal (41)
FirstGoalscorer (42)
LastGoalscorer (43)
PlayerToScore (44)
FirstHalfHandicap (45)
FirstHalfTotal (46)
SetBetting (47)
GroupBetting (48)
MatchplaySingle (49)
MatchplayFourball (50)
MatchplayFoursome (51)
TiedMatch (52)
TopBatsman (53)
InningsRuns (54)
TotalTries (55)
TotalPoints (56)
FrameBetting (57)
ToScoreFirst (58)
ToScoreLast (59)
FirstScoringPlay (60)
LastScoringPlay (61)
HighestScoringQtr (62)
RunLine (63)
RoundBetting (64)
LineBetting (65)

MoneyAmount

The amount of money, not explicitly including an indication of the currency concerned (as that can be inferred from the users' session). This is a decimal number to 2 places of decimal.

OrderActionType

Enumeration defining the type of an order history audit record. Domain values are:

Placed (1)
ExplicitlyUpdated (2)
Matched (3)
CancelledExplicitly (4)
CancelledByReset (5)
CancelledOnInRunning (6)
Expired (7)
MatchedPortionRepricedByR4 (8)
UnmatchedPortionRepricedByR4 (9)
UnmatchedPortionCancelledByWithdrawal (10)
Voided (11)
Settled (12)
Suspended (13)
Unsuspended (14)
ExpiredByMatching (15)
Unsettled (16)
Unmatched (17)
MatchedPortionRepriced (18)
CreatedFromLightweightPrice (19)
CancelledOnComplete (20)

OrderFillType

Enumeration defining the matching behaviour desired for an order. Domain values are:

Normal (1)
After the initial attempt is made to match this order any unmatched portion of the order is to remain as an unmatched order capable of being matched at a subsequent time.

FillAndKill (2)
After the initial attempt is made to match this order any unmatched portion of the order is immediately cancelled. Thus the order can be partially matched but there will never be any unmatched portion of the order remaining.

FillOrKill (3)
On the initial attempt to match this order if it is not possible to match a specified amount of the order then none of the order will be matched. If it had been possible to match at least the specified amount then the amount that can be matched will be matched and the remaining unmatched amount will be cancelled.

FillOrKillDontCancel (4)
On the initial attempt to match this order if it is not possible to match a specified amount of the order then none of the order will be matched. If it had been possible to match at least the specified amount then the amount that can be matched will be matched and the remaining unmatched amount will be not be cancelled but left as an unmatched order.

SPIfUnmatched (5)
Same as *Normal* but any unmatched portion of the order is to be matched at SP when the market is turned in-running (or completed).

OrderStatus

Enumeration defining the status of an order. Domain values are:

Unmatched (1)
The order is active and has some amount available for matching (the order may be partially matched).

Matched (2)
The order has not been settled and it does not have any unmatched amount. Either the order was fully matched or it was partially matched and then cancelled.

Cancelled (3)
This order has been cancelled and at least some of the order was unmatched at the time of expiration..

Settled (4)
The order has been settled.

Void (5)
The order has been voided.

Suspended (6)
At least some of this order is unmatched but the order is suspended and is not available for matching..

Percentage

Decimal containing a value that is to be interpreted as a percentage. For example the value 89.5642 is to be interpreted as meaning 89.5642 per cent.

Polarity

Enumeration defining whether an order is for or against a line. Domain values are:

For (1)
This Order is being issued on the 'for' side of the Selection (also known as the 'left' side or the 'back' side).

Against (2)
This Order is being issued on the 'against' side of the Selection (also known as the 'right' side or the 'lay' side).

PostingCategory

Enumeration defining the categories of a posting. Domain values are:

Settlement (1)
This posting resulted from the settlement, unsettlement or resettlement of a specific order.

Commission (2)
This posting resulting from the charging of commission on market settlement, unsettlement or resettlement.

Other (3)
This posting resulted from any other cause (for example, a lodgement or withdrawal).

PunterCommissionBasis

Enumeration defining the basis on which commission for the punter is to be calculated. Domain values are:

NetMarketWinnings (1)
Commission is to be calculated on the basis of a Punter's net winnings in a market.

MatchedForSideStakeExcludingPush (2)
Commission is to be calculated on the basis of matched for side amount of each order (regardless of whether the polarity of the order is for or against). However (for all market types other than EachWayNonHandicap, EachWayHandicap and EachWayTournament) the

amount of commission charged is to be reduced by the voidPercentage specified at settlement. For example if the void percentage was 100% (push) no commission should be charged, whereas if the voidPercentage is 50% the amount of commission charged should be halved. voidPercentage should be ignored for the each-way market types.

MatchedForSideStakeIncludingPush (3)

Commission is to be calculated on the basis of matched for side amount of each order (regardless of whether the polarity of the order is for or against). Commission is to be charged if the net settlement figure is zero, unless the market or selection is voided.

MatchedAgainstSideStakeExclusingPush (4)

Commission is to be calculated on the basis of matched against side amount of each order (regardless of whether the polarity of the order is for or against). However (for all market types other than EachWayNonHandicap, EachWayHandicap and EachWayTournament) the amount of commission charged is to be reduced by the voidPercentage specified at settlement. For example if the void percentage was 100% (push) no commission should be charged, whereas if the voidPercentage is 50% the amount of commission charged should be halved. voidPercentage should be ignored for the each-way market types.

MatchedAgainstSideStakeIncludingPush (5)

Commission is to be calculated on the basis of matched against side amount of each order (regardless of whether the polarity of the order is for or against). Commission is to be charged if the net settlement figure is zero, unless the market or selection is voided.

MatchedRiskedStakeExcludingPush (6)

Commission is to be calculated on the basis of matched amount risked on each order. That is, if the polarity of an order is 'For' the relevant stake is the matched for side stake and if the polarity is 'Against' the relevant stake is the matched against side stake. However (for all market types other than EachWayNonHandicap, EachWayHandicap and EachWayTournament) the amount of commission charged is to be reduced by the voidPercentage specified at settlement. For example if the void percentage was 100% (push) no commission should be charged, whereas if the voidPercentage is 50% the amount of commission charged should be halved. voidPercentage should be ignored for the each-way market types.

MatchedRiskedStakeIncludingPush (7)

Commission is to be calculated on the basis of matched amount risked on each order. That is, if the polarity of an order is 'For' the relevant stake is the matched for side stake and if the polarity is 'Against' the relevant stake is the matched against side stake. Commission is to be charged if the net settlement figure is zero, unless the market or selection is voided

Price

Price expressed as decimal odds. This is expressed to 12 decimal places of decimals. However only exact values that are on our odds ladder (with points for decimal, fractional and moneyline) will be accepted.

PriceFormat

Enumeration defining the format of a price. Domain values are:

Decimal (1)
: The price is expressed in decimal format – in particular the price is the decimal representation of the payout for 1 currency unit stake.

Fractional (2)
: The price is expressed in fractional format – in particular the fraction is the winnings for 1 currency unit stake.

American (3)
: The price is expressed in American format – in particular if the price is greater than 0 it means the amount of winnings for 100 currency unit stake whereas if the price is less than zero it means the amount of currency unit that needs to be staked to win 100 currency units.

SelectionStatus

Enumeration defining the current status of a Selection. Domain values are:

Inactive (1)
: The Selection is not active and has never had any Orders issued against it.

Active (2)
: The Selection is active (that is, Orders can be issued against it).

Suspended (3)
: Orders can not currently be placed on this Selection.

Withdrawn (4)
: The Entrant explicitly referenced by the Selection has withdrawn from the Event and so Orders can no longer be placed on this Selection.

BallotedOut (9)
: The Entrant explicitly referenced by the Selection has been balloted-out.

Voided (5)
: Orders can no longer be placed on this Selection and Orders previously placed for or against the Selection have been voided.

Completed (6)
: The Selection is completed. No further Orders can be issued against the Selection but the result of the Selection is either not yet known or has not yet been entered.

Settled (8)
: This Selection has already been settled. Individual selections can be settled in advance of other selections in the market being settled (early settlement).

short

String

Timestamp

Time expressed in UTC. All times in the system are expressed in UTC.

TradeType

Enumeration defining whether a trade was a back or a lay. Domain

Formatted

values are:

Back (1)

The backer took a lay price that was available.

Lay (2)

The layer took a back price that was available.

WithdrawRepriceOption

Enumeration defining the action to take on a specific Order if a withdrawal occurs on the Market that could cause a Rule-4 deduction factor to be applied to the Order (this option only controls what happens to the unmatched parts of Orders, those parts that have already been matched will have the rule-4 deduction applied regardless of the value of this option). Domain values are:

Reprice (1)

Reprice the unmatched parts of the Order. It is anticipated that this would be the usual option specified by Layers.

Cancel (2)

Cancel the unmatched parts of the Order.

DontReprice (3)

Do not reprice the unmatched parts of the Order.

Return Codes

RC000 Success
The API call was processed successfully

RC001 ResourceError
The API call was not processed successfully because of some critical resource constraint. This error was not caused by invalid or incorrect parameters specified on the API call, but rather as a result of a serious resource constraint within the exchange. Examples of serious resource constraint include insufficient memory, disk space and operating system resources.

RC002 SystemError
The API call was not processed successfully because of some serious technical error within the system. This error was not caused by invalid or incorrect parameters specified on the API call, but rather as a result of a serious technical configuration error within the system itself.

RC005 EventClassifierDoesNotExist
An Event Classifier with the handle specified does not exist. Specify the handle of an existing Event Classifier and retry.

RC008 MarketDoesNotExist
A Market with the handle specified does not exist. Specify the handle of an existing Market and retry.

RC011 SelectionDoesNotExist
A Selection with the handle specified does not exist. Specify the handle of an existing Selection and retry.

RC015 MarketNotActive
The action requested cannot be performed because the Market specified is not active. Activate the Market and retry.

RC016 MarketNeitherSuspendedNorActive
The action requested cannot be performed because the Market specified is neither suspended nor active. Either suspend or activate the Market and retry.

RC017 SelectionNotActive
The action requested cannot be performed because the Selection specified is not active. Activate the Selection and retry.

RC019 InsufficientVirtualPunterFunds
The Order specified could not be placed because the increase in the amount of the Virtual Punter's funds that would need to be frozen as a result of that Order is greater than the value specified for 'maxVirtualReservationIncrease'.

RC021 OrderDoesNotExist
An Order with the handle specified does not exist. Specify the handle of an existing Order and retry.

RC022 NoUnmatchedAmount
The Order specified could not be placed, cancelled or changed because the amount requested is negative or the entire stake of the Order has already been matched.

RC114 ResetHasOccurred
The order was not placed because the expectedSelectionResetCount specified on the Order does not match the current selectionResetCount for the Selection.

RC127 OrderAlreadySuspended
The Order specified is already suspended.

RC128 TradingCurrentlySuspended
The Order could not be processed because all trading is currently suspended on the Exchange.

RC131 InvalidOdds
The odds specified are not valid. There are a number of reasons why the odds specified are not considered valid including (1) the value specified is less than or equal to 1.0, (2) the odds specified is not one of the odds values on the odds ladder (which is a combination of normal decimal, fractional and moneyline odds) and (3) the odds and stake specified mean that it is would not be possible to match the order.

An example of this third case is if on a against order you specify an amount of 0.01 and a price of 1.2. The amount of layers liability involved in this case would be 0.002 which would round to zero and therefore be an invalid match.

RC136 WithdrawalSequenceNumberIsInvalid
The Order was not placed because the withdrawal sequence number specified is greater than the current withdrawal sequence number for the market.

RC137 MaximumInputRecordsExceeded
APIs that accept a variable number of iunput parameters have a limit on the number of parameters that can be specified, for implementation and denial of service reasons. The limit depends on the API concerned. That limit was exceeded in this case.

RC208 PunterSuspended
The requested action can not be performed because the Punter concerned is currently suspended.

RC240 PunterProhibitedFromPlacingOrders
The Order was not placed / changed or cancelled because the Punter is either:

1. explicitly prohibited from placing Orders (and that includes changing or cancelling Orders) or
2. is explicitly prohibited from placing Order in advance of his identity being established and his identity has not yet been established.

RC241 InsufficientPunterFunds
The Order was not placed or changed because the Punter does not have sufficient unfrozen funds to cover the increase in maximum downside that would result for the Order being placed or changed.

RC271 OrderAPIInProgress
This API was not executed because another order API is currently in progress for this Punter. A Punter can not issue more than one order API (PlaceSimgleOrder, PlaceGroupOrder or ChangeOrder) at the same time.

RC274 PunterOrderMismatch
The API could not complete because one or more of the Orders specified were not issued by the Punter specified or there is a mis-match between the Punter implied and the object specified.

RC281 MarketNotEnabledForMultiples
The market of one or more selections specified does not support multiple bets.

RC285 MultipleLayerParameterAlreadyExists
A MultiplePriceMultipler for the MultipleLayer specified with the numberOfSelections specified already exists.

RC288 LevelsRequestedExceedsMaximum
The number of levels of combination requested exceeds the system defined maximum. Specify a smaller value for number of levels and re-try.

RC289 NoMultipleOfferAvailable
The requested multiple bet was not matched because there was no multiple offers available on the requested selection at the price requested.

RC293 InRunningDelayInEffect
An attempt was made to (i) change an order that is currently subject to an in-running delay or (ii) place a multiple order or to get a multiple quotation involving a selection that is currently in-running.

RC295 MultipleSelectionsUnderSameEvent
An attempt was made to place a multiple bet that contained two or more Selections belonging to markets under the same event classifier. A multiple bet can not contain more thn one selection in the same event.

RC296 MultipleSelectionsWithSameName
An attempt was made to place a multiple bet that contained two or more selections with exactly the same name. In general, a multiple bet is not allowed to contain two selections with the same name (although selections in markets of type MacthOdds or OverUnder are ignored for this condition).

RC299 DuplicateOrderSpecified
The same order was specified more than once in the same API call. Ensure that an order is only specified once in the API call and re-try.

RC301 OrderNotSuspended
The order specified is not currently suspended.

RC302 PunterIsSuspendedFromTrading
The requested operation could not be completed because the Punter concerned is currently suspended from trading.

RC303 PunterHasActiveOrders
An attempt was made to unsuspend Punter from trading. A Punter can only be unsuspended if the Punter currently has no active orders. However the Punter concerned currently does have active orders.

RC304 PunterNotSuspendedFromTrading
The requested operation could not be completed because the Punter concerned is not currently suspended from trading.

RC305 ExpiryTimeInThePast
The requested operation could not be performed because the expiry time specified is in the past. Retry the operation either specifing an expiry time in the future or not specifying an expiry time at all.

RC306 NoChangeSpecified
The requested operation (change order) could not be performed because the price specified is the current price of the order concerned.

RC307 SoapHeaderNotSupplied
The SOAP header was not specified. All External API calls must include a SOAP header.

RC308 IncorrectVersionNumber
An incorrect version number specified in API header.

RC309 NoUsernameSpecified
You must specify your username in the API header.

RC310 InvalidParameters

Invalid parameters were passed to the web method (for example, user sends a null parameter when it's not optional).

RC311 NoPasswordSpecified
You must specify your password in the API header.

RC312 MultipleCombinationExclusionAlreadyExists
There is currently an active MultipleCombinationExclusion with the set of Selections specified existing for the multiple layer specified.

RC313 MultipleCombinationExlcusionDoesNotExist
There is not currently an active MultipleCombinationExclusion with the set of Selections specified for the multiple layer specified.

RC405 InvalidPassword
The password specified is not a valid password. Specifically it does not conform to the rules defined for such passwords.

RC406 PunterIsBlacklisted
The operation requested was not executed because the Punter concerned is currently black-listed from performing that action.

RC425 PunterNotRegisteredAsMultipleLayer
The requested action could not be performed because the Punter concerned has not been registered as a multiple layer.

RC462 PunterAlreadyRegisteredForHeartbeat
The operation could not complete because the Punter concerned is already registered for a Heartbeat.

RC463 PunterNotRegisteredForHeartbeat
The operation could not complete because the Punter concerened is not registered for a Heartbeat. In addition to the application not having registered a Heartberat this situation can also arise in the unlike event that a system component was reset, in which case the application must reregister the Heartbeat.

RC473 ThresholdSpecifiedTooSmall
The threshold value specified is less than the system defined minimum threshold value.

RC477 UnmatchedOrderCouldResult
The attempt to place an order was not successful because it could result in an unmatched order remaining on the system. The specific API call used can only be used for orders that can not result in a an unmatched order. For example, you can not specify a KillType of *FillOrKillDontCancel.*

RC533 PunterNotAuthorisedForAPI
The punter concerned is not authorised to use the external API.

RC597 MarketIsForRealMoney
The requested operation cannot be performed because the market concerned is not a play market but the currency specified is a play currency.

RC598 MarketIsForPlayMoney
The requested operation cannot be performed because the market concerned is a plat market but the currency specified is a real currency.

RC892 CannotChangeToSPIfUnmatched
The requested operation could not be performed because the orderFillType of the order concerned is not Normal.